QUIZ

Fill in the method name:

```
def ???(self, value):
  if self.has_space():
    item_to_add = Node(value)
    if self.is_empty():
      self.head = item_to_add
      self.tail = item_to_add
    else:
      self.tail.set_next_node(item)
      self.tail = item_to_add
    self.size += 1
  else:
    print("Sorry, no more room!")
```

- dequeue
- peek
- has_space
- enqueue

👏 `enqueue()` adds a new item with the passed in value to the end of the queue if there is enough space.

Why is `.has_space()` a useful method for the `Queue` class to have?

- It allows us to see if we can `.enqueue()` a new value on the end of the queue.

👏 `.has_space()` tells us if there is space for us to `.enqueue()` — or add — an additional value to the end of the queue.

- It allows us to see if the queue is empty.
- It allows us to see if we can `.peek()` the first item on the queue.
- It allows us to see if we can `.dequeue()` the first item on the queue.

If you instantiate an empty queue, what should `self.head` and `self.tail` be set to?

`head` and `None`

`None` and `tail`

`None` and `None`

If you are instantiating an empty queue, there will be no head or tail nodes in the queue yet.

`head` and `tail`

What should be the `if` statement for `.has_space()`?

```
def has_space(self):
  if ???:
    return self.max_size > self.get_size()
  else:
    return True
```

`self.is_empty()`

`self.size > 0`

`self.get_size()`

`self.max_size`

If the queue has a `max_size` then it should check if the `max_size` is greater than its size. If there is no `max_size`, then there is space for an additional item.

Fill in the method name:

```
def ???(self):
  if self.get_size() > 0:
    item_to_remove = self.head
    if self.get_size() == 1:
      self.head = None
      self.tail = None
    else:
      self.head = self.head.get_next_node()
    self.size -= 1
    return item_to_remove.get_value()
  else:
    print("This queue is totally empty.")
```

`enqueue`

`has_space`

`dequeue`

👏 `dequeue()` checks to see if the queue is empty and if not, this method removes the head item from a queue if there is an item to remove.

Take a look at the following code based on the `Queue` class you've been working with. What do we know is true for `muffins_to_be_eaten`?

```
muffins_to_be_eaten = Queue(10)

muffins_to_be_eaten.enqueue("blueberry")
muffins_to_be_eaten.enqueue("corn")
muffins_to_be_eaten.peek()
muffins_to_be_eaten.dequeue()
```

`muffins_to_be_eaten.max_size == muffins_to_be_eaten.size`

`muffins_to_be_eaten.head == muffins_to_be_eaten.tail`

👏 There is only one node remaining, making that node both the `head` and `tail` of the queue.

Which values go where in `Queue`'s `__init__()` method?

```
def __init__(self, max_size=None):
  self.size = #1
  self.head = #2
  self.tail = #3
  self.max_size = #4
```

#1: 0, #2: None, #3: None, #4: max_size

When a queue is instantiated there is no head or tail node yet, and because it is empty its size is 0.

#1: size, #2: head, #3: tail, #4: max_size

#1: size, #2: head, #3: None, #4: max_size

Fill in the blank:

`is_empty()` is a useful method for preventing _____.

queue overflow

queue underflow

Queue underflow may occur when we try to remove items from a queue when there are no items to remove.

What will the return value be if we add a final statement calling `.dequeue()` on `sharks_in_the_shark_tank`?

```
sharks_in_the_shark_tank = Queue(5)

sharks_in_the_shark_tank.enqueue("Laura")
sharks_in_the_shark_tank.enqueue("Dhruti")
sharks_in_the_shark_tank.dequeue()
sharks_in_the_shark_tank.enqueue("Kenny")
```

"Dhruti"

`.dequeue()` should return the value of the first node on the queue.

A node with a value of "Laura"

"Kenny"

Which two `Queue` methods should alert if the queue is empty when called?

`.__init__()` and `.peek()`

`.__init__()` and `.enqueue()`

`.dequeue()` and `.peek()`

If the queue is empty, then it is not possible to remove or view the value of the first item.

`.enqueue()` and `.dequeue()`